Evolving Robot Vision: Increasing Performance through Shaping

Simon Perkins
Los Alamos National Laboratory
Mail Drop D436
Los Alamos, NM 87545
s.perkins@ed.ac.uk

Abstract- Automated methods for designing robot controllers based on machine-learning techniques have shown great promise when applied to simple robot tasks, but in order to 'scale up' to more complicated problems they will require assistance from human experts, a process that is often called 'robot shaping'. In this paper, the difficult problem of learning how to visually track moving objects is examined. It is shown that through the use of shaping techniques, this intractable learning problem can be made soluble. Controllers are evolved in simulation and then transferred to a real robot.

1 Introduction

1.1 Robot Learning for Visual Tasks

In recent years a lot of research effort has been put into producing real-world robot controllers automatically using techniques from the machine learning and optimization research fields. Engineering robot controllers by hand is difficult for many reasons, including problems of sensor and actuator noise, calibration errors, sensitivity of the robot-environment system to small changes in starting conditions, the requirement for real-time behaviour, and the sheer difficulty of humans 'thinking down' to the robot's level. The idea of deriving controllers for complex tasks automatically from the robot's own interaction with its environment is therefore an attractive one.

While the field has produced many interesting results, much work in robot learning is subject to the criticism that the tasks tackled are usually relatively simple. Typically the robots are equipped with only a few low-bandwidth sensors and actuators and are used in tasks such as obstacle avoidance and light-following which are robust to small inaccuracies in behaviour.

Equipping robots with vision opens up a whole field of more challenging tasks, and introduces a number of new problems to be overcome by a learning system. Visual sensors typically produce enormous amounts of raw data at a very high rate, and this data is generally difficult to relate unambiguously to the physical structure of the scene in front of the robot. A number of researchers have successfully used robot learning for visual tasks. Asada et al. (1996), for instance, have used Q-learning to train visually equipped mobile robots to shoot tennis balls through a 'goal'. Harvey et al. (1994) have evolved a neural network controller that allows a gantry robot to find coloured targets using vision. Finally, Jakobi (1998) has evolved a neural net motion-tracking controller for a one degree of freedom robot head.

1.2 Shaping: Lending a Hand

Any learning system has its limits. In general, the more flexible and general purpose the learning algorithm, and the fewer the assumptions made about the nature of the problem, the harder it is for the learning system to find a solution in a reasonable time. Many interesting visual tasks are probably beyond the capabilities of pure general-purpose learning algorithms and it seems clear that for really complicated tasks the learning system must be assisted by a human designer. In robot learning, giving assistance in this way is often called 'shaping' (Dorigo and Colombetti, 1993, 1998). There are several different categories of shaping method, but three of the most important are:

Controller decomposition Controllers for complex tasks can often be broken down into a hierarchy of smaller modules. It is often much easier to train these individual modules separately or sequentially, than to train the whole controller at once. In robotics, controllers are often decomposed in a behaviour-based way, with some modules performing simple sub-tasks, and others coordinating the activation of those modules (e.g. Mahadevan and Connell, 1992; Dorigo and Colombetti, 1993).

Progressive problem difficulty One important problem faced by learning robots is the difficulty of 'getting off the ground'. At the start of the learning process the robot will probably behave in a relatively random fashion, and in some training scenarios this might mean that the robot gets very little useful feedback on how to behave correctly. This problem can sometimes be alleviated by initially training the robot on easier versions of the full task. Asada et al. (1996) call this 'learning from easy missions'.

A fuller taxonomy of shaping methods can be found in Perkins (1999b); Perkins and Hayes (1996). For more discussion of robot shaping in general see Dorigo and Colombetti (1998).

2 Motion Tracking

2.1 Task Description

The 'robot' used in the experiments described here is actually a robot 'head', equipped with a single camera that can

be moved independently in 'pan' and 'tilt' directions. Such robot heads are increasingly common in research laboratories around the world, and probably the most common task for which they are used is the task of motion-tracking. We define this task roughly as 'keeping the camera pointed at a single smallish moving object in an otherwise stationary scene'.

Almost all existing motion-tracking controllers have been carefully programmed by hand. Our goal is to try to *learn* how to do this task, based only on scalar evaluations, and then to see if shaping techniques can be used to make learning the task easier. The only other work we are aware of that tackles the same task is that of Jakobi (1998). He has evolved neural network controllers that are capable of 1-D motion-tracking (controlling just the pan axis), but the raw video input is heavily pre-processed to make learning the task tractable.[1] In our work we use less initial pre-processing, and shaping techniques are used instead to make the task tractable. The principal motivation behind our preference for shaping is that shaping techniques often require only a high-level analysis of the task (e.g. a top-down decomposition), whereas sensor preprocessing usually relies on a low-level understanding of the problem. Perkins (1999b) provides more discussion of this approach.

2.2 A Simulator for Evolving Real-World Trackers

Evolving controllers for the motion-tracking task would be a tedious process if carried out directly on the real robot. Although the camera can move quite rapidly, at around $100\,^{\circ}s^{-1}$, the evolutionary runs presented in this paper would still take several weeks each to run. The robot head would be unlikely to survive this much continuous operation! As a result the controllers are initially evolved in simulation before being transferred to the real robot. Evolving in simulation has the very useful additional benefit that we can provide the learning system with a much more 'informed' evaluation mechanism than is possible in the real world.

Our simulator is designed using a methodology called the 'radical envelope of noise' (Jakobi, 1997), which can be summarized as:

- For environmental features that are *relevant* to the task, model them as well as possible, and add a small amount of noise to mask modelling inaccuracies.
- For environmental features that are *irrelevant* to the task, model them poorly, and add huge amounts of noise to prevent the learning system paying any attention to those features.

In accordance with this philosophy, a visual simulator for the motion-tracking task was developed. For this task we are not really interested in the details of the background behind the target to be tracked, and so this can be replaced with a highly variable randomly generated one. The moving target is coloured in a similar way. Some care was taken to ensure that the simulated background shows random variation at both fine and coarse scales in a similar fashion to the real world. Figure 1 shows a typical simulated scenes generated for the motion-tracking task.

The image dimensions are 320×240 pixels. All pixels vary in intensity between 0 and 255. Multiplicative Gaussian noise with $\mu = 1.0, \sigma = 0.02$ is used to corrupt pixel intensity values slightly. The target varies in radius between 4° and 10°, and moves in the simulated scene with a velocity between 5 and $20^{\circ}s^{-1}$, and the simulator runs on a 4Hz cycle.

It is not necessary to model the background scene in a realistic way. However, it *is* necessary to model the way in which the target moves fairly accurately. This was achieved by basing the imaging model on empirical measurements of known scenes taken using the real robot head. Building a simulator using empirical data has previously been suggested by Miglino et al. (1995). The stepper motors on the robot head actually allow almost error-free positioning, but the velocity signals sent to the simulator by the controller are corrupted by adding multiplicative Gaussian noise with $\mu = 1.0, \sigma = 0.02$ to mask image modelling inaccuracies.

3 Evolutionary Details

3.1 The TAG Architecture

Many different learning architectures could be used for learning visual tasks. For reasons of generality, flexibility, and ease of use however, we use a variant of genetic programming (Koza, 1992) called TAG. The TAG architecture is described fully in Perkins (1999b), but a brief overview of its essential features is given here.

In contrast to standard GP, the structures evolved by TAG are general acyclic graphs rather than trees. Evolving graphs rather than trees allows controllers that use values computed by sub-graphs more than once to be be expressed in a more compact and evolvable way than is possible with simple trees. Several other authors have suggested adding graphs into GP for similar reasons (Poli, 1996; Teller and Veloso, 1996). TAG graphs are initially created by constructing a 'bag' of randomly chosen GP-style function and terminal nodes. The number of nodes in the bag is chosen randomly within a certain range: 10–20 nodes per bag in these experiments. TAG graphs can have more than one output value (for instance one output per actuator), and for each required output, a node is selected from the bag at random to return that value. If the node is not a terminal node, then further nodes are connected to its inputs as necessary and this connection process continues in recursive fashion until there are no nodes in the graph that need input connections. Nodes may connect to other nodes that already form part of the graph, but loops are prevented by insisting that no node may connect to a node that

[1] The image is first cropped to leave a narrow horizontal strip of interest. This is then sub-sampled to produce a 32×1 pixel image. Frame differencing and thresholding then reduce this to a binary image showing the location of 'rapid' image change.

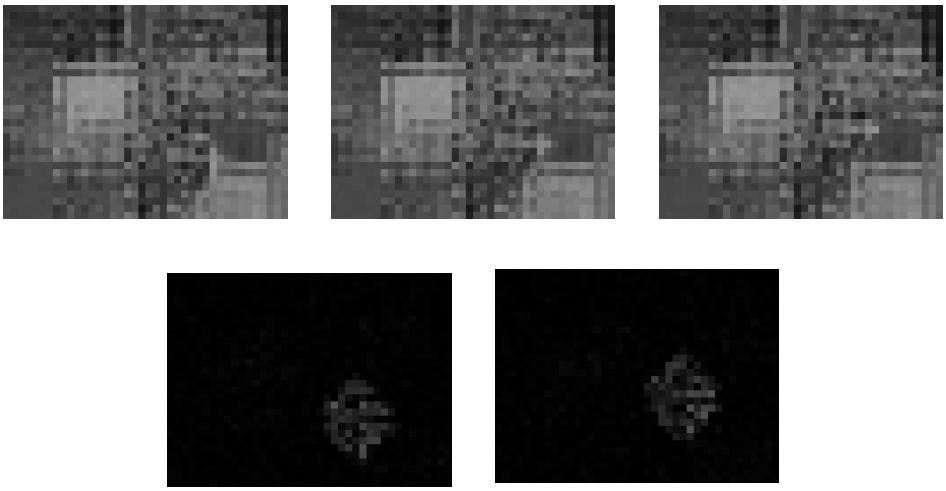

Figure 1: Typical simulator image sequence for the motion-tracking task. The top row of images shows a sequence of three frames produced by the simulator with the camera stationary. The moving target is very difficult to spot in these images, but can be seen more clearly in the bottom row of images, which shows the absolute difference between images 1 and 2, and between images 2 and 3. Note that the images are sub-sampled.

is its ancestor (i.e. its parent, or an ancestor of its parent). In order to ensure that the process terminates it is necessary that the bag contains at least one terminal node to start with. At the end of the connection process, not all nodes are necessarily in use — these unused nodes act as spare genetic material or 'introns' that may be used later.

Traditional GP uses sub-tree crossover as its primary genetic operator. TAG also uses crossover, but the operator must be modified for use with graphs. For the most part, 'sub-graph crossover' is very similar to sub-tree crossover. Starting with two parents, two offspring graphs are created by copying. Then, a node is selected in each offspring to act as an exchange node. To perform crossover, the exchange nodes are simply swapped between the two graphs, together with all their descendent nodes (a node is a descendent of another node if it is a child of that node or a descendent of a child).

The tricky part of sub-graph crossover is deciding what to do about connections that previously connected into the exchanged sub-graphs (other than connections to the exchange nodes themselves). One obvious answer is just to randomly reassign such connections, but TAG tries to take a more intelligent approach that attempts to preserve structure where possible. In brief, every node is associated with a fixed 'tag' value that is uniquely assigned when that node is first created. When a node is copied, the copy is given the same tag value. If a particular sub-graph turns out to be a useful component, then copies of it will tend to multiply in the population. Each of those copies will have similar sets of nodes and associated tags. When TAG is reassigning a connection into an exchanged region of nodes, it first checks to see if there are any nodes present in the newly formed individual that have the same tag as the node that the connection was previously connected to. If there is, then a connection is made to that node. If not, then the connection is reassigned randomly. The key idea is that it is less destructive if connections are reassigned to structures that are similar to the ones they were previously connected to.

Offspring can also be produced by mutation. In this case, a single child is initially generated by copying a single parent. Then, R nodes are selected at random for mutation, where R is Poisson-distributed with expected value 2.[2] Some types of node have internal parameters that are 'micro-mutatable', and if such a node is selected, then 90% of the time, one of its parameters is mutated slightly. In all other cases the node has one of its input connections reassigned randomly (if applicable), or is itself replaced with a random node.

TAG has a number of other interesting features, principally the use of a 'rational allocation of trials' (RAT) mechanism (Teller and Andre, 1997) for reducing fitness evaluation time, but space precludes a fuller description here. See Perkins (1999b) for details.

Apart from the aforementioned exceptions, TAG is a relatively conventional evolutionary algorithm. It is a 'steady-state' algorithm in that as soon as offspring are generated they are put back into the evolving population — there is no concept of a generation. TAG maintains a population of size N. Parents are selected using tournament selection: at each evolutionary step, M individuals are chosen from the population at random and their fitnesses are compared. The fittest individual in the tournament is chosen as one parent, and one of the remaining individuals is chosen randomly as the other parent. 50% of the time the two parents are bred using crossover. Only one of the two potential children is actually generated. The other 50% of the time, mutation is used to derive a single offspring from the fitter of the two parents. In either case the offspring replaces the less fit of its parents. This evolutionary

[2] If $R = 0$, then R is re-generated.

cycle is repeated a total of T times during a single run. In the experiments reported here, $N = 100$, $M = 4$ and usually, $T = 25000$.

3.2 Shaping and Tag

Shaping generally implies an incremental acquisition of behaviour with newly learned skills building upon previously acquired skills. The human trainer must design an incremental path of increasing competence that the robot is to follow. Each 'stage' along this path or 'shaping regime', results in the acquisition of another unit of competence in some area related to the overall task. The controller must not forget skills it has already learned, even if they are not directly relevant to the current stage of the training process, and it must be able to combine previously learned skills together to form new, more complex skills.

The shaping experiments reported here use three basic frameworks for incremental learning. Note that in practice, different frameworks may be used at different stages of a shaping regime.

Single Agent (SA) In the simplest cases, no modification is made to the basic learning system. At each stage of the shaping regime, the designer adjusts the evaluation function and/or environmental constraints as appropriate, and the evolving TAG population is simply required to adapt to the modified task as best it can. No provision for preserving previously learned skills is made, so this technique is only suitable for regimes where the task does not change qualitatively from one shaping stage to the next.

Multiple Agent, Fixed Interaction (MAFI) In the next most complicated case, we develop a controller that consist of a collection of 'agents', each agent being a single evolved TAG graph, and each being evolved during a single stage of the shaping regime. At the end of each evolutionary stage, the best agent in the evolving population is 'frozen' and added into the current controller. In the MAFI framework, the designer ensures that there is no conflict between different agents, or hand-designs an arbitration system that allows the agents to work together.

Multiple Agent, Learned Interaction (MALI) Finally we can consider the case where agents evolved at different stages may conflict with each other. In this case we require that the agents learn how to cooperate. We use a simple method called 'hierarchical evolutionary gating' (HEG). In the HEG framework, agents that might conflict possess a special additional real-valued output called the 'validity'. For an agent to control an actuator, its validity must be greater than zero. If there is more than one such agent, then the most recently evolved agent wins. If no agents are eligible, the actuator assumes a default value for the current cycle. The validity mechanism allows an agent to take control if it needs to override previously evolved agents, and to relinquish control to previously evolved agents if that is not necessary.

4 Shaping Experiments

4.1 Simulation Runs

In other published work (Perkins, 1999a,b) the benefits of shaping using progressive problem difficulty and controller decomposition for a simpler light-tracking task were demonstrated. Here we apply similar techniques to the harder motion-tracking task.

4.1.1 Control Runs

Before the shaped experiment, we carried out two control experiments, to see if the task could be learned without shaping. The first control experiment attempts to evolve a monolithic controller for a 1-D version of the motion tracking task in which only the pan axis needs to be controlled. The evolved controllers consist of single TAG graphs with a single output controlling the camera pan speed. In the second control experiment we again attempt to evolve a controller without shaping. This time the full 2-D motion tracking task is used, and the evolved controllers consist of single TAG graphs, each with two outputs: one controlling the camera pan speed, one controlling the camera tilt speed.

Standard GP-style function nodes and terminals were used for both control experiments, consisting of the standard arithmetic functions: $+$, $-$, $\times$, $\div$, $>$, `neg`, `sgn` and `abs`; plus the ephemeral random constant $\Re$. $\div$ returns 1.0 if its second argument is zero. $>$ returns 1.0 if its first argument is greater than its second argument, or 0.0 otherwise. `sgn` returns 1.0 if its single argument is greater than zero, or -1.0 otherwise. Random constants are initialized to values between -1.0 and 1.0.

In addition to these familiar functions three additional terminal nodes and three additional functions were introduced for the tracking task. `vis2` provides access to the camera sensor. It contains five internal parameters that define a rectangular 'receptive field' in the image. Four parameters define this rectangle, and the fifth defines a sampling resolution within that receptive field. `vis2` is unusual in that it returns a 2-D *array* of values which correspond to the *absolute change* in intensity value in each of the pixels in the defined receptive field. The arithmetic binary functions described above are designed to cope with arrays in a sensible fashion. If one input is scalar and the other is an array, then an array of the same size as the input array is returned, with each value being the result of applying the binary function to the scalar and the corresponding value in the input array. If both inputs are arrays, then the two arrays are first registered with their centres as close as possible, and the largest common sub-array is extracted from each array. The output is then an array of the same size in which each value is the result of applying the

binary function to the corresponding values in the two input arrays. In conjunction with the `vis2` terminal node, three simple image processing function nodes are defined: `avg`, which returns the mean value of an array; `mx`, which returns the first x-moment of the array, and `my`, which returns the first y-moment of the array. These functions simply return their input if passed scalar values. Finally, two terminals to read the current camera pan and tilt velocity: `panspd` and `tiltspd`.

Note that the `vis2` terminal performs frame differencing automatically. It can also sample the image at range of resolutions, from 2×2 pixels per array element, to 32×32 pixels per element. This pre-processing was found to be necessary to make the problem soluble, even with the shaping that was carried out. Processing arrays of values is computationally demanding, so the `vis2` terminal is prevented from returning arrays with more than 150 elements. This turned out to make the problem considerably harder than expected.

The fitness function for the motion-tracking task is defined in terms of the tracking error E:

$$F = -\sum_{t}^{n}(1-\gamma^t)E_t \qquad (1)$$

where E_t is the tracking error at time-step t, defined as the angular offset between the camera 'straight ahead' direction and the target direction. For the 1-D control experiment, only the horizontal component of this angle is considered. γ is a weighting constant, set to 0.5 in these experiments, and n is the number of time-steps for which the trial is continued — in these experiments $n = 10$. Note that the sum is negated to ensure that higher fitness values are better.

Each evolutionary run lated 50000 tournaments, and each of the two control experiment was carried out 50 times with different random number seeds to get good statistics. Figure 2 shows the performance curves derived from these runs. The graphs show how the tracking error of the best individual in the population varied throughout the run. The solid curve shows the median best error over the 50 trials, and the dotted curves show the 10th and 90th percentile best errors. The graphs show quite clearly that the best tracking error does not change from its initial value — the tracking problem is too hard for the evolutionary system to solve, unassisted.

4.1.2 Shaping Runs

To see whether shaping could be used to make the motion-tracking problem tractable, a ten-stage shaping regime was tested. The essential idea behind the regime is to get the controller to first learn how to locate the target in the image, and only then to learn how to track the target. This is an example of controller decomposition. The regime also trains the controller to control pan and tilt axes separately, and uses progressive problem difficulty where appropriate in an attempt to simplify learning.

For the target-location stages, the controller is required to signal its estimate of the position of the target by setting two 'memory units' to values representing the horizontal and vertical offset of the target relative to the centre of the image. Fitness for these stages is then simply the difference between the memory unit value and the correct value, in degrees, negated so that higher fitness values are better. Each fitness trial lasts for two robot cycles (0.5 sec), at the end of which the memory units are examined. The same functions and terminals as used in the control experiments are available to all evolving agents, except where noted below. The full regime looks like this:

1 1-D target-location task with targets appearing only in the leftmost half of the image. The random background texture is enhanced to make targets 'easier' to spot. Controllers consist of a single agent with an output targeting the horizontal offset memory unit.

2 As stage 1, except that the texture enhancement is removed. The same population of single-agent controllers continues to evolve (SA framework).

3 1-D target-location task with targets appearing anywhere in the image. The random background texture is enhanced as before. The best agent from stage 2 is frozen and incorporated into the controller. New evolving agents have one output targeting the horizontal offset memory unit and a validity output for conflict resolution (MALI framework).

4 As stage 3, except that the texture enhancement is removed. The evolving population of second agents continues to evolve (SA framework).

5 1-D motion-tracking task with targets appearing anywhere in the image. The best agent from stage 4 is frozen and incorporated into the controller. New evolving agents have a single output controlling the pan speed (MAFI framework). They also have access to a new terminal `mem0` that can read the horizontal offset memory unit.

6–9 As stages 1–4 except that training is on the tilt axis version of the 1-D target-location task. The first three evolved agents are temporarily disabled and two more agents are evolved for tilt-axis target location.

10 All previously evolved agents are enabled, and a sixth agent is evolved on the full 2-D motion-tracking task. New evolving agents have a single output controlling the tilt speed and have access to a new terminal `mem1` that can read the vertical offset memory unit.

Each stage is 25000 tournaments long. Each stage is repeated 20 times with different random number seeds and the best result from each stage is used as the precursor for 20 runs of the next stage.

For comparison, a hard-wired motion-tracking controller was also created. This controller has access to the same raw

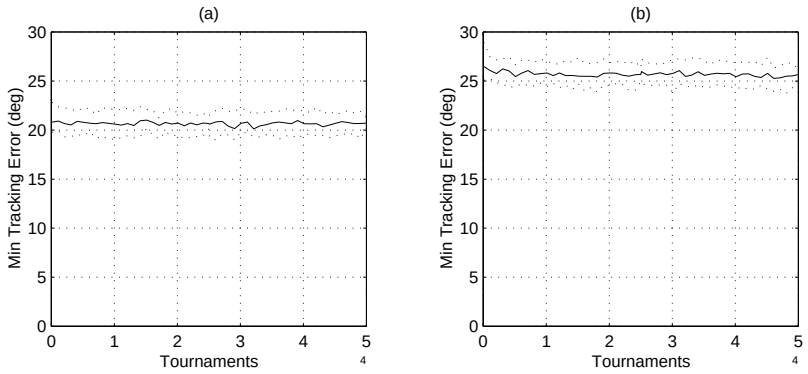

Figure 2: Performance graphs for the control motion-tracking experiments. (a) 1-D motion-tracking task. (b) 2-D motion-tracking task.

information as the evolved controller and makes use of similar functional blocks. It's basic strategy is to hold the camera stationary every other frame in order to highlight the moving target in the difference input image. The centroid of the region of motion is then located, and appropriate signals sent to the pan and tilt motors in order to bring this region into the centre of the image.

Table 1 shows the end-of-run performances from all the 2-D tracking experiments. The 'Error' column shows the mean tracking error of the best individual seen from each experiment. The hit rate indicates what percentage of the time the controller kept the target in view for a whole trial. The 'Adj. Error' column gives the mean tracking error of the best individual on just those cases for which the target was kept in view.

Controller	Error	Hit Rate	Adj. Error
Unshaped	25.5 ± 0.6	65.1%	20.3 ± 0.6
Shaped	16.1 ± 0.5	90.1%	14.3 ± 0.3
Hand-coded	1.6 ± 0.1	99.8%	1.5 ± 0.03

Table 1: Final performance comparison of the motion-tracking controllers. All errors given in degrees.

The table shows clearly that the shaped evolved controller outperforms the unshaped evolved controller, but is itself outperformed by the hand-coded controller. This result is analyzed in detail in Section 5.

4.2 Back to Reality

The shaped evolved controller appears to be able to perform motion-tracking at least some of the time in simulation, but how does its ability transfer to the real robot?

To answer this question, the shaped controller and and the hand-coded controller were both transferred to a real robot head (the one upon which the simulation was based) and tested in various ways. To get a simple objective comparison, a toy car racetrack was used. A single car was set to circle a track at a fairly constant speed and the robot head was positioned nearby. To make the car more visible a small balloon was attached to the car. The size and speed of the car and balloon in the image were carefully chosen to match the parameters of the simulation. Both the evolved and hand-coded controllers were able to track the car indefinitely. Figure 3 shows a typical sequence of frames from a robot's eye view.

Table 2 shows the mean tracking error for each controller over a typical period of 25 cycles. The error was determined by hand-locating the approximate centre of the moving target in each scene and determining the distance to the image centre. As the table suggests, the evolved and hand-coded controllers performed comparably on this task.

Controller	Error
Shaped	11.8 ± 1.1
Hand-coded	11.0 ± 0.9

Table 2: Comparison of real-world performances of evolved and hand-made motion-tracking controllers.

A subjective comparison between the two controllers in a variety of other testing environments was also carried out. Unfortunately here the results were much less impressive. As soon as the testing situation diverged from the training situation, e.g. larger or smaller targets, or multiple moving objects, then the evolved controller failed completely. The hand-coded controller on the other hand performed fairly robustly in these other situations and by-and-large was able to track the main moving object.

5 Analysis and Conclusions

Other work by the author (Perkins, 1999a,b) has shown that shaping can be used to produce a controller for a light-tracking task that competes very favourably with a hand-coded controller. The current work demonstrates that motion-tracking controllers evolved using similar techniques fare less well in a comparison with hand-coded controllers. The principal rea-

Figure 3: Typical motion-tracking sequences for the evolved and hand-made controllers.

son for this is that the problem is much harder. Without shaping, the best solution that TAG could find involved just keeping the camera stationary all the time. In comparison, on the light-tracking task, even unshaped evolved controllers were able to track lights some of the time. For motion-tracking the major problem is that any camera motion induces large scale image motion which swamps any image motion caused by the moving target. This problem does not exist for light-tracking. The successful evolved controller learned to hold the camera stationary every other frame — the same strategy as used by the hand-coded controller. This was achieved despite the fact that the body of the evolved controller is entirely reactive. [3]

The subjective real-world comparison also reveals a potential danger of simulation-to-reality transfer, even when carefully designed simulators are used. The evolved controller worked reasonably well while the testing environment was similar to the training environment, but it extrapolated very unintuitively to unseen situations. In contrast, the hand-coded controller, which was designed with an eye to likely extrapolations, behaved well in a wide variety of situations.

On the positive side, the results provide further evidence that this form of shaping (progressive problem difficulty and controller decomposition) can help an evolutionary system develop robot controllers that are impossible to evolve without shaping. It has been shown before (e.g. Dorigo and Colombetti, 1993) that controller decomposition with different modules communicating in a 'control flow' way can accelerate robot learning. Our research goes further in showing that controller decomposition with 'data flow' interaction is also useful.

Acknowledgements

This work was carried out while the author was at the Department of Artificial Intelligence, University of Edinburgh, Scotland.

Bibliography

Asada, M., Noda, S., Tawaratsumida, S., and Hosoda, K. (1996). Purposive behaviour acquisition for a real robot by vision-based reinforcement learning. *Machine Learning*, 23:279–303.

Dorigo, M. and Colombetti, M. (1993). Robot shaping: Developing situated agents through learning. Technical Report TR-92-040, International Computer Science Institute, Berkley, CA 94704.

Dorigo, M. and Colombetti, M. (1998). *Robot Shaping: An Experiment in Behaviour Engineering*. MIT Press/Bradford Books.

Harvey, I., Husbands, P., and Cliff, D. (1994). Seeing the light: Artificial evolution, real vision. In Cliff, D., Meyer, J.-A., and Wilson, S., editors, *From Animals to Animats 3: Proc. 3rd Int. Conf. Simulation of Adaptive Behavior*. MIT Press.

Jakobi, N. (1997). Evolutionary robotics and the radical envelope of noise hypothesis. *Adaptive Behavior*, 6:325–368.

Jakobi, N. (1998). Evolving motion-tracking behaviour for a panning camera head. In Pfeifer, R., Blumberg, B., Meyer, J.-A., and Wilson, S., editors, *From Animals to Animats 5: Proc. 5th Int. Conf. Simulation of Adaptive Behavior*. MIT Press.

Koza, J. R. (1992). *Genetic programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.

Mahadevan, S. and Connell, J. (1992). Automatic programming of behaviour-based robots using reinforcement learning. *Artificial Intelligence*, 55:311–365.

Miglino, O., Lund, H., and Nolfi, S. (1995). Evolving mobile robots in simulated and real environments. *Artifical Life*, 2(4):417–434.

Perkins, S. (1999a). Evolving effective visual tracking through shaping. In W. Banzhaf et al., editor, *GECCO 99: Proc. Genetic and Evolutionary Computation Conference*, San Francisco, CA. Morgan Kaufmann. July 13–17, Orlando, Florida.

Perkins, S. (1999b). *Incremental Acquisition of Complex Visual Behaviour using Genetic Programming and Robot Shaping*. PhD thesis, Dept. of Artificial Intelligence, University of Edinburgh, 5 Forrest Hill, Edinburgh, Scotland. Available from http://www.dai.ed.ac.uk/students/simonpe/pubs.html.

Perkins, S. and Hayes, G. (1996). Robot shaping — principles, methods and architectures. Technical Report 795, Dept. of Artificial Intelligence, Univ. of Edinburgh, Scotland.

Poli, R. (1996). Parallel distributed genetic programming. Technical Report CSRP-96-15, School of Computer Science, University of Birmingham, UK.

Teller, A. and Andre, D. (1997). Automatically choosing the number of fitness cases: The rational allocation of trials. In Koza et al., editor, *Proc. Genetic Programming '97*. Morgan Kaufmann.

Teller, A. and Veloso, M. (1996). PADO: Learning tree-structured algorithms for orchestration into an object recognition system. Technical report, Department of Computer Science, CMU, Pittsburgh, PA.

[3] The two state cycle is in fact achieved by monitoring the `panspd` and `tiltspd` inputs which reflect the previous cycle's speed commands.